# Universal Manipulation Policy Network for Articulated Objects

Zhenjia Xu, Zhanpeng He, and Shuran Song, *Member, IEEE*

***Abstract*—We introduce the Universal Manipulation Policy Network (UMPNet) – a single image-based policy network that infers closed-loop action sequences for manipulating articulated objects. To infer a wide range of action trajectories, the policy supports 6DoF action representation and varying trajectory length. To handle a diverse set of objects, the policy learns from objects with different articulation structures and generalizes to unseen objects or categories. The policy is trained with self-guided exploration without any human demonstrations, scripted policy, or pre-defined goal conditions. To support effective multi-step interaction, we introduce a novel Arrow-of-Time action attribute that indicates whether an action will change the object state back to the past or forward into the future. With the Arrow-of-Time inference at each interaction step, the learned policy is able to select actions that consistently lead towards or away from a given state, thereby, enabling both effective state exploration and goal-conditioned manipulation.**

***Index Terms*—Deep learning in grasping and manipulation, perception for grasping and manipulation.**

## I. Introduction

THE ability to effectively interact and manipulate unknown articulated objects is critical for many robotics tasks. However, due to the large variance in the objects' kinematic structure and 3D geometry, the actual action trajectories can vary drastically across different object instances and categories. Fig. 1 shows examples of action trajectories conditioned on different objects for opening a door, turning a switch, or opening a drawer. Extensive prior works have studied how to manually design or learn an object-specific policy for each type of interaction (e.g., opening doors). However, such policies are often time-consuming to design and fail to generalize across objects with different articulation structures.

While these interaction sequences are drastically different in their low-level geometric trajectories, many of them can be summarized by a similar high-level function conditioned on the objects' underlying geometric and kinematic structure. For example, the motion trajectory of a door opening can be represented by a function conditioned on its frame size and its rotation axis, and a similar function can also be used for opening a fridge, a microwave, or even a laptop. By learning to interact with a diverse set of articulated objects, the system is able to acquire a generalizable knowledge about objects' articulation structure and how these structures would react to different actions. Such knowledge goes beyond a specific object instance or category, allowing a universal interaction policy for any articulated objects.

Can we enable a robot to automatically acquire these basic concepts about the object structure through self-supervised interactions and use them to infer the corresponding manipulation policies? In this paper, we introduce the **Universal Manipulation Policy Network (UMPNet)** – a single policy network that discovers possible manipulation policies for an articulated object from visual observations (i.e., RGB-D images). The action trajectories inferred by the policy network (shown in Fig. 1) highlight the following attributes:

- *General action representation:* In order to model *all* possible actions for *any* articulated object, the network should be able to represent a general action space with little constraints – it should be able to represent continuous actions in SE(3) with arbitrary trajectory length. To achieve this goal, we formulate an action trajectory by its initial 3D position and a sequence of action directions, which allows the network to describe complex motion trajectories with varying sequence lengths.
- *Closed-loop action sequence:* Instead of predicting a single step action (e.g., push or pull), we are interested in predicting long-horizon sequential actions that could describe a complex motion trajectory. However, due to error accumulation and partial observation, directly predicting the full trajectory from the initial state can be challenging. To address this issue, we use a closed-loop formulation where the network continues to predict the next action conditioned on the object's initial and current state, allowing the network to adjust its action prediction based on its continuous visual observation of the object.
- *Arrow-of-Time awareness:* Most of the action trajectories are bi-directional in time (i.e., they are valid in either direction). Hence, conditioning on a single state can result in multiple effective next actions that would change the object's state with the same magnitude. However, to avoid the back-and-forth actions, the network takes the history state as input and infers an additional "Arrow-of-Time (AoT)" attribute for each action. This AoT label indicates

Manuscript received September 9, 2021; accepted December 31, 2021. Date of publication January 13, 2022; date of current version January 28, 2022. This letter was recommended for publication by Associate Editor B. Calli and Editor M. Vincze upon evaluation of the reviewers' comments. This work was supported in part by the National Science Foundation under Grant CMMI-2037101, and in part by the Amazon Research Award. *(Corresponding author: Zhenjia Xu.)*

The authors are with the Columbia University, New York, NY 10027 USA (e-mail: xuzhenjia1997@gmail.com; zh2405@columbia.edu; shurans@cs.columbia.edu).

This letter has supplementary downloadable material available at https://doi.org/10.1109/LRA.2022.3142397, provided by the authors.

Digital Object Identifier 10.1109/LRA.2022.3142397

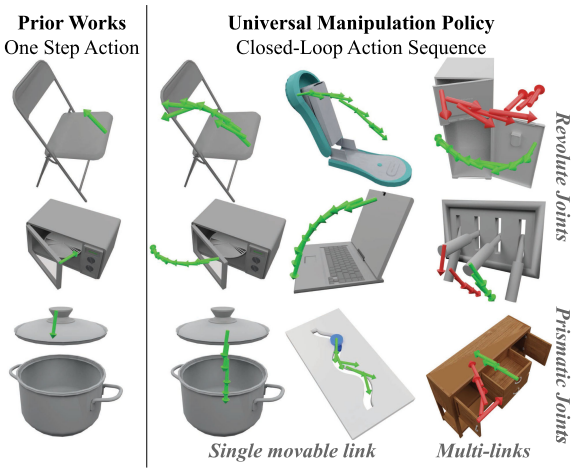


Fig. 1. **Universal Manipulation Policy for Articulated Objects.** Instead of predicting a single step action, UMPNet predicts complex closed-loop 6DoF action sequences with varying trajectory length. As a result, the same policy network is able to handle a diverse set of objects regardless their joint types or number of links.

whether this action will change the object state back to the past or forward into the future. Apart from encouraging exploring new states, this Arrow-of-Time inference also allows us to *directly* apply the network in "goal conditioned manipulation," where we can simply swap out the initial state with the goal state and choose the actions using a reversed Arrow-of-Time.

In summary, we present a unified framework that discovers possible manipulation policies for an articulated object from visual observations. By using self-guided exploration, the policy network is able to learn a wide range of action trajectories for a diverse set of objects and generalize to unseen objects and categories. The training does not require any human demonstrations or pre-defined goal conditions. We validate our approach on two manipulation tasks (1) open-ended state exploration and (2) goal-conditioned manipulation. The experiments demonstrate that UMPNet is able to outperform alternative approaches in both tasks significantly.

## II. Related Work

*Open-loop manipulation with pose estimation:* Many works have focused on learning task-specific manipulation primitives, such as grasping [1], pushing [2] and tossing [3]. For articulated objects, methods have focused on handling doors, and drawers [4]–[12]. These prior works typically start with object pose estimation [13], [14] and then use the object pose to compute an open-loop motion trajectory. However, the action trajectory designed for one task (e.g., opening doors) may be too specific to be applied to other objects or tasks (e.g., pushing button). Moreover, performing pose estimation for articulated objects with *unknown category and kinematic structure* is an extremely challenging task. On the contrary, our model does not require any object detection, pose estimation or part segmentation, and demonstrates that it is in fact not *necessary* to perform explicit pose estimation to perform effective manipulations.

*Learning action trajectories from demonstrations:* Another popular method for robots to acquire new manipulation skills

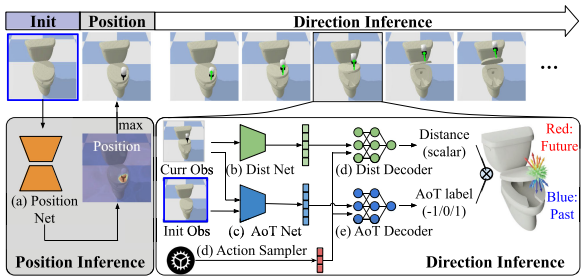


Fig. 2. **Approach overview.** UMPNet takes visual observation (i.e., RGB-D images) of an articulated object as input and generates a sequence of actions in SE(3) space to explore novel object states. **(Left)** A grasp position is selected in the first interaction step. **(Right)** In following steps, the outcomes for each action candidates ($r_{\text{dist}}$ and $r_{\text{AoT}}$) are inferred and then used for action direction selection. $r_{\text{dist}}(a_t^{\text{dir}})$ infers the potential moving distance of the joint after applying the action $a_t^{\text{dir}}$. $r_{\text{AoT}}(a_t^{\text{dir}})$ infers whether or not the action will move the object toward a novel state. The action direction with largest $r_{\text{dist}}$ and positive $r_{\text{AoT}}$ will be selected.

is learning from demonstrations. This approach has been explored extensively in reinforcement learning literature [15]. Researchers has tried using behavioral cloning to learn from human demonstration data captured by various methods, for example, motion capture [16], [17], videos [18]–[20] and virtual reality [21], [22]. However, these works requires collection of large amount of high quality demonstrations with action and pose annotation, which is expensive to obtain. In contrast, our framework generates its own training data by allowing the agent to actively interact with objects and explore the environment.

*Single-step action affordance:* Action affordance describes the possibility of an action to be applied to a given location in the environment. The task of affordance prediction does not limit to a specific kind of object or action primitive. Building on the well-studied image segmentation problems, many existing methods have been developed to learn object affordance through passive observations, such as learning human-object interaction hotspots from video [23], [24] and contact heatmap from RGB-D image [25]. The work most related to ours is "Where2Act" by Mo *et al.* [26], where the algorithm can infer single step action affordance for different articulated objects. However, limited by its single step formulation, this approach fails to generated long-horizon motion trajectories for goal-conditioned manipulation tasks, which is the focus of our approach.

## III. Approach

The goal of the manipulation policy $\pi$ is to generate a sequence of actions to interact with a random articulated object which would result in novel states that haven't been visited before. Taking Fig. 2 as an example, to effectively explore novel states of the object (i.e., a toilet), the algorithm should be able to (a) choose the right position on the object to interact with (i.e., interacting with the cover instead of the base), (b) select a proper action direction (i.e., pulling up instead of pushing down), and (c) consistently select actions in the following steps to explore novel states (i.e., keeping pulling up the cover instead of moving up-and-down). These three requirements directly correspond to the three key components of our algorithm, which are action position selection (a), action distance (b) and Arrow-of-Time

inference (c) for action direction selection. As a result, the final system is able to learn through a self-guided exploration process, without explicit human demonstrations [22], scripted policy [26], or pre-defined goal conditions [27].

### A. Problem Formulation

The task is defined as follows: given a visual observation of an articulated object in the form of an RGB-D image at the initial and current state $o_0, o_t \in \mathbb{R}^{W\times H\times 4}$, the agent with a policy $\pi$ generates an action $a_t$ at each step $\pi(o_t, o_0) \rightarrow a_t$ that satisfies the aforementioned requirements. The action is represented in SE(3) space, parameterized by end-effector (i.e., a suction-based gripper) position and moving direction $a_t = (a_t^{\text{pos}}, a_t^{\text{dir}})$, where $a_t^{\text{pos}} \in \mathbb{R}^3$ is a 3D coordinate and $a_t^{\text{dir}} \in \mathbb{R}^3, (||a_t^{\text{dir}}|| = 1)$ is a unit vector in 3D indicating the end-effector moving direction.

In the first interaction step, the policy selects a 3D position $a_0^{\text{pos}}$ to apply action (i.e., an immobilizing grasp via suction). To execute the action, the agent moves its end-effector to this position, with an orientation perpendicular to the object surface. Note that the gripper orientation (determined by the surface normal) can be different from the action direction $a_t^{\text{dir}}$ (determined by the Direction Inference Networks Section III-C). In each following step, the agent will select a 3D direction $a_t^{\text{dir}}$ and move its end-effect 0.18(m) along that direction, the position $a_t^{\text{pos}}$ is fixed relative to the objects surface. The suction behavior is implemented as a force constraint between the suction cup and the selected 3D position on the object. The orientation of the end-effector is always aligned with the surface normal during the interaction.

### B. Position Inference

To start, the policy needs to determine a suitable position on the object 3D surface $a_0^{\text{pos}}$ to apply action (i.e., a immobilizing grasp via suction). To do so, the algorithm needs to select a pixel from the observation image $o_0$ to apply action. The selected pixel will then be projected back to the 3D space using the depth value provided in the RGB-D image.

We formulate this problem as an image labeling task, where the position network (Fig. 2(a)) takes in an RGB-D image and predicts per-pixel position affordance score $P \in [0, 1]^{W\times H}$. The affordance score $P(w, h)$ implies the likelihood of the object part movement when applying an action in this position. We use a U-Net architecture for this task, the network is supervised by the outcome of the executed action (one out of $W \times H$ pixels). The ground truth label is 1 if and only if the object state is changed in any of the future steps. The network is trained with Binary Cross-Entropy loss.

Note that simply selecting a position belonging to a movable link is a necessary but not sufficient criteria. For example, if the selected position is very close to the joint axis, the agent will not be able to apply enough force to move the object part. Furthermore, the label is affected by the quality of direction selection. A correct position can still be labeled as a negative case if the object state is not changed due to wrong direction predictions in the following steps.

### C. Direction Inference

At this point, the end-effector has grasped the object link at $a_0^{\text{pos}}$ which is visible to the camera. Conditioned on this information, the policy then needs to select a 3D direction $a_t^{\text{dir}}$. To select the action direction, the algorithm need to first sample a set of action candidates, and evaluate each action candidate's effectiveness. The "effectiveness" is measured by the moving distance of the object joint position $r_{\text{dist}}(a_t^{\text{dir}})$ and Arrow-of-Time attribute $r_{\text{AoT}}(a_t^{\text{dir}})$, defined as following:

$$r_{\text{dist}}(a_t^{\text{dir}}) = ||\vec{j}_t - \vec{j}_{t-1}||$$

$$\gamma = (\vec{j}_t - \vec{j}_{t-1}) \cdot (\vec{j}_{t-1} - \vec{j}_0)$$

$$r_{\text{AoT}}(a_t^{\text{dir}}) = \begin{cases} -1 \; r_{\text{dist}}(a_t^{\text{dir}}) > \delta \quad \& \quad \gamma < 0 \\ \;\; 0 \; r_{\text{dist}}(a_t^{\text{dir}}) \leq \delta \\ \;\; 1 \; r_{\text{dist}}(a_t^{\text{dir}}) > \delta \quad \& \quad \gamma \geq 0 \end{cases}$$

where $\vec{j}_t$ is the object joint state in each step $t$ and $\delta$ is a threshold to determine whether the state is effectively changed. $\delta$ is 0.15 m for prismatic joint and 8.6° for revolute joint. The following paragraph provides details on how to generate action candidates $\{a_t^{\text{dir}}\}$, and infer $r_{\text{dist}}(a_t^{\text{dir}})$ and $r_{\text{AoT}}(a_t^{\text{dir}})$.

To generate direction candidates $\{\hat{a}^{\text{dir}}\}$, one naive method would be uniformly sampling in the SO(3) space. However, limited by the number of samples, the sampled directions can only cover a small portion of the continuous action space that does not include the optimal directions. To address this issue, we use a heuristic approach, iterative cross-entropy method (CEM), to reduce the sampling space to achieve efficient direction sampling. The algorithm starts with uniform sampling the SO(3) space for $N$ samples. Then, it evaluates the sampled actions based on the predicted action scores: $s(\hat{a}) = \tilde{r}_{\text{dist}}(\hat{a}^{\text{dir}}) \cdot \tilde{r}_{\text{AoT}}(\hat{a}^{\text{dir}})$. In the next iteration, the algorithm re-samples the candidates with probability correlated to its score: $p(\hat{a}) \propto e^{T*s(\hat{a})}$, where $T = 20$ is a temperature value. Added a random noise, they are considered as candidates in the second interaction. In this way, the samples in the second iteration will concentrate on the region that has more "potential," leading to better performance with the same number of samples. Detailed comparisons are listed in appendix. Our final model uses CEM sampling with 64 samples.

To infer the moving distance $\tilde{r}_{\text{dist}}(a_t^{\text{dir}})$ for an action candidate, the network needs to consider the object's current state and grasp position which are both encoded in the current observation $o_t$. Taking in the RGB-D image of the current state, DistNet (Fig. 2(b)) outputs embedding vector $\psi(o_t)$. Then DistDecoder (Fig. 2(d)) takes both embedding vector $\psi(o_t)$ and action $a$ as input, and outputs a scalar as the distance prediction $\tilde{r}_{\text{dist}}(a_t^{\text{dir}})$. DistNet is a convolution neural network and the output is flattened to an embedding vector. Dist-Decoder is a fully-connected neural network trained using MSE loss $\mathcal{L}_{dist}$ for the executed action $a_t$.

Different from $\tilde{r}_{\text{dist}}(a_t^{\text{dir}})$ inference, Arrow-of-Time $\tilde{r}_{\text{AoT}}(a_t^{\text{dir}})$ inference is conditioned on on both current observation and initial observation. For single-step interaction, any action that changes the object's state would result in a novel state. However, it is not true for multi-step interactions – the policy can move the object link back-and-forth without

exploring any new states. To address this issue, we proposes an "Arrow-of-Time" (AoT) action attribute that indicates whether the action will change the object state back to the initial state or forward into the future (i.e., a novel state). Specifically, AoTNet (Fig. 2(c)) takes the current and initial observation as input and outputs another embedding vector $\phi(o_t, o_0)$. This embedding vector is then combined with the action embedding to infer the final AoT label for this action $\tilde{r}_{\text{AoT}}(a_t^{\text{dir}})$. The network architectures of the AoT branch is similar to those of the Dist branch while the only differences are the different input dimensions of the Dist Net and the AoT Net as well as the different output dimensions of the AoT Decoder and the Dist Decoder. The model is trained as a three-way classification with Cross-Entropy loss $\mathcal{L}_{AoT}$. The final loss for direction inference is: $\mathcal{L} = \lambda \mathcal{L}_{dist} + \mathcal{L}_{AoT}$, where $\lambda = 100$ in our experiments.

### D. Training

All training data come from interaction trials executed by the policy trained from scratch. A FIFO replay buffer (size=6400) is used to store training data. To collect data with both positive and negative AoT labels, we employ contradictory policy for direction inference within a sequence. In the first half of each sequence, we select action with positive AoT prediction for execution to move the object away from its initial state. In the second half, actions with negative AoT prediction are executed to encourage the object to move back. 16 trajectories are collected in each epoch. The sequence length is 4 at the beginning. After 1000 epochs, it increases by 2 every 400 epochs, until reaching 20. $\epsilon$-greedy is used during training, where $\epsilon$ decreases linearly from 1 to $\epsilon_{min}$ within $n$ epochs. In position inference, $n = 300$ and $\epsilon_{min} = 0.1$. In direction inference, $n = 500$ and $\epsilon_{min} = 0.2$.

Position module and direction module are trained with 8 iterations accordingly in each epoch. In each position training iteration, we sample a batch (size=16) of examples from the replay buffer with a 1:1 positive to negative ratio. In each direction training iteration, 1:1:1 samples from positive, negative, and not-moving data form a batch (size=24).

### E. Goal Conditioned Manipulation With Reversed AoT

While open-ended interaction is useful for exploring and collecting information about the environment, most manipulation tasks are goal conditioned – the policy needs to generate actions that would lead toward a given goal state instead of a random novel state. Although the policy is trained with only open-ended exploration, the learned policy can be directly applied to perform goal conditioned manipulation without additional training.

The key idea for performing the goal-conditioned task is to swap out the initial observation with the goal state observation as the input to the policy. Then by executing the actions with a reversed Arrow-of-Time (i.e., negative AoT), the policy tries to move object back to the "past," which will effectively move the objects toward the goal. If the AoT prediction of all direction candidates are non-negative (no blue arrows in Fig. 3), the trajectory will terminate.

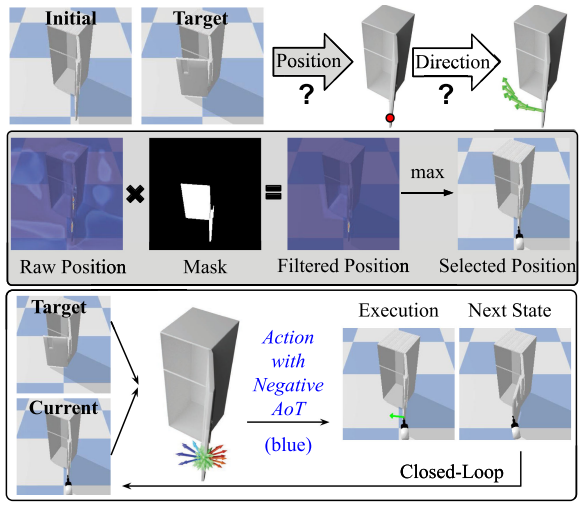


Fig. 3. Goal conditioned manipulation.

Apart from choosing the right action direction, another unique challenge for goal-conditioned manipulation is how to choose the correct link to interact when there are multiple movable links on the object (e.g., fridge with double doors in Fig. 3). While the position heatmap predicted by the network covers all movable links, only interacting with the right one can lead to the goal. Therefore, to choose a proper position, we first compute a difference mask between the initial and target observation. Then, we multiply the raw position heatmap and the mask to get the filtered position affordance (remove the pixels that are not changed). The final position is selected from the filtered heatmap. The algorithm for goal-conditioned manipulation is illustrated in Fig. 3.

## IV. Evaluation

Our simulation environment uses objects from PartNet-Mobility [28] and physics engine from Pybullet [29]. We use 12 categories for training and 10 categories for testing. There are 504 training object instances, 132 testing object instances from training categories, and 261 object instances in the testing categories. We randomly load an articulated object into the simulation for each interaction session with a randomly initialized pose and joint configurations.

### A. Open-Ended State Exploration

We first evaluate UMPNet's effectiveness in exploring novel states of an articulated object. Being able to effectively explore the possible states of an object without a specific goal is a critical first step for many robot learning algorithms since it is often used to collect the initial observation about the environment to initiate the training. While random explorations can be used for simple environments, they are often not sufficient for tasks involving high-dimensional action space, where the majority of the actions will not change the object joint state in a meaningful way.

Instead, an *effective* state exploration policy should be able to choose actions that can (1) significantly change the joint state of an object and (2) lead to novel states that have not been visited

TABLE I
EFFECTIVE STATE EXPLORATION[1]

| | Novel instances in training categories | | | | | | | | | | | | Testing categories | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Where2Act | 0.94 | **2.08** | 1.10 | **0.79** | **0.92** | 1.24 | 1.05 | **1.06** | 0.80 | **0.74** | **0.96** | 0.57 | 0.96 | 1.48 | 1.01 | 1.17 | 1.17 | **1.95** | **0.82** | 1.02 | 1.38 | 0.81 |
| AoTOnly | 0.99 | 1.42 | 1.05 | 0.63 | 0.62 | 1.01 | 0.76 | 0.62 | 0.61 | 0.54 | 0.57 | 0.51 | 0.75 | 1.10 | 1.06 | 1.10 | 1.14 | 1.46 | 0.49 | 0.86 | 1.21 | 0.80 |
| SignedDist | 0.84 | 1.68 | 1.04 | 0.53 | 0.91 | 1.25 | 1.23 | 0.69 | 0.73 | 0.43 | 0.65 | 0.51 | 0.75 | 1.10 | 1.06 | 1.10 | 1.14 | 1.46 | 0.49 | 0.86 | 1.21 | 0.80 |
| UMPNet | **1.02** | **2.08** | **1.37** | 0.73 | **0.92** | **1.29** | **1.26** | 1.03 | **0.81** | 0.70 | 0.90 | **0.66** | **1.10** | **1.50** | **1.14** | **1.18** | **1.32** | 1.87 | 0.77 | **1.05** | **1.69** | **0.90** |
| Single action effects ↑ | | | | | | | | | | | | | | | | | | | | | | |
| Where2Act | 0.38 | 0.45 | 0.34 | 0.25 | 0.52 | 0.56 | 0.49 | 0.56 | 0.45 | 0.50 | 0.58 | 0.26 | 0.39 | 0.39 | 0.45 | 0.42 | 0.51 | 0.53 | 0.50 | 0.66 | 0.24 | 0.34 |
| Where2Act+HP | **0.72** | 0.85 | 0.89 | 0.48 | 0.60 | 0.83 | **0.85** | 0.72 | 0.62 | **0.63** | 0.73 | 0.50 | 0.75 | 0.87 | **0.79** | 0.84 | 0.81 | 0.89 | 0.54 | 0.86 | 0.91 | 0.65 |
| SingleStep | 0.31 | 0.42 | 0.39 | 0.26 | 0.47 | 0.51 | 0.48 | 0.49 | 0.44 | 0.47 | 0.57 | 0.24 | 0.44 | 0.38 | 0.39 | 0.41 | 0.45 | 0.45 | 0.47 | 0.78 | 0.29 | 0.31 |
| AoTOnly | 0.58 | 0.77 | 0.69 | 0.42 | 0.47 | 0.68 | 0.62 | 0.67 | 0.50 | 0.44 | 0.59 | 0.44 | 0.70 | 0.76 | 0.65 | 0.82 | 0.61 | 0.81 | 0.44 | 0.80 | 0.83 | 0.50 |
| SignedDist | 0.43 | 0.59 | 0.66 | 0.38 | 0.47 | 0.54 | 0.58 | 0.58 | 0.46 | 0.38 | 0.48 | 0.38 | 0.60 | 0.57 | 0.51 | 0.58 | 0.57 | 0.65 | 0.36 | 0.55 | 0.68 | 0.47 |
| UMPNet | 0.70 | 0.85 | 0.90 | 0.52 | 0.60 | 0.87 | 0.81 | 0.74 | 0.64 | 0.55 | 0.74 | 0.52 | 0.77 | 0.85 | 0.76 | 0.85 | 0.80 | 0.92 | 0.56 | 0.86 | 0.93 | 0.68 |
| UMPNet+HP | 0.71 | **0.86** | **0.90** | **0.57** | **0.64** | **0.88** | 0.83 | **0.74** | **0.65** | 0.60 | **0.74** | **0.55** | **0.77** | **0.88** | 0.78 | **0.86** | **0.83** | **0.92** | **0.56** | **0.88** | **0.93** | **0.70** |
| Ratio of unique states visited ↑ | | | | | | | | | | | | | | | | | | | | | | |

before. The first property requires the system to understand the object structure, and the second property requires the system to be aware of the interaction history.

*Metrics:* We use two metrics to evaluate the effectiveness of state exploration: (1) Single action effects – measures the joint state difference before and after each interaction step $D = ||\vec{j}_t - \vec{j}_{t-1}||/\delta$. The threshold of significant state change $\delta$ is 0.15 m for prismatic joint and 8.6° for revolute joint.

This metric evaluates whether the algorithm can choose the action that would change the state of the object most significantly. (2) Novel state visited – measures the ratio between the number of unique states visited among all interaction steps: $\text{ratio} = \#\text{unique\_states}/\#\text{steps}$. Two states consider the "same" when the object's joint difference is less than $\delta$. This metric evaluates whether the algorithm is aware of the interaction history and chooses the action leading to novel states that have not been visited before.

*Algorithm comparisons:* We compare our final model with the following alternative approaches:

- Where2Act [26]: This algorithm takes the current observation as input and selects single-step action. The model is with binary-classification loss where the action is positive if only the moving distance is larger than a threshold.
- Where2Act+HP: an additional heuristic that filters out actions that has a larger than 90° angle with last-step action. This heuristic helps to avoid back-and-forth actions, however cannot be applied for goal-conditioned manipulation.
- SingleStep: Single-step version of our method that only takes the current observation as input.
- AoTOnly: This method only outputs AoT label for each action without the distance inference.
- SignedDist: Instead of inference AoT and distance as separate outputs, this method infers signed distance by multiplying the AoT and distance value $r_{\text{singed}} = r_{\text{AoT}} \cdot r_{\text{dist}}$

*Results and analysis:* Quantities and qualitative results are summarised in Table I and Fig. 4.

Effect of the AoT prediction: Both [ Where2Act ] and [ SingleStep ] only take the current observation as input and infer actions for one step; hence, they do not need to understand the interaction history. From Table I we can see that [ Where2Act ] is able to achieve similar performance in "single action effects," however, both [ Where2Act ] and [ SingleStep ] cannot effectively explore novel states with more interaction steps. Since both algorithms are not aware of interaction history, we observe that the policy often selects actions that would manipulate the object link back-and-forth instead of exploring new possible object states. When combined with the heuristic the algorithm [ Where2Act+HP ] can avoid back-and-forth action, however, it is sensitive to error propagation, where one sub-optimal action would affect all following steps through the filtering process, results in worse performance. Fig. 4 shows examples of action prediction results for [ UMPNet ]. With just the Arrow-of-Time prediction, [ UMPNet ] is able to identify the actions that would always move the object from the past states (i.e., red arrows); therefore, it is able to visit novel states much more frequently. When combined with heuristic filter, the performance improves slightly.

Effect of the distance prediction: Compared to [AoTOnly], we can observe that by explicitly predicting the distance value for each action candidate, [UMPNet] can better differentiate

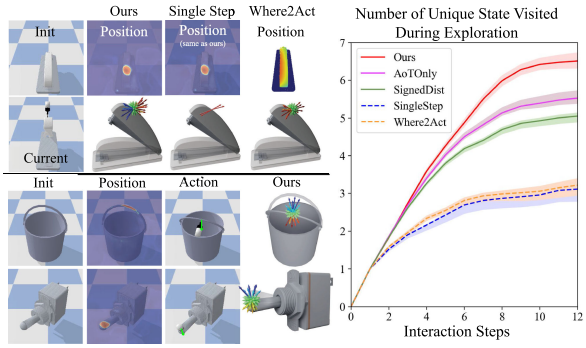


Fig. 4. **Open-ended state exploration.** Arrow length indicates the inferred distance value, color indicates the inferred AoT label. We visualized the uniform samples to better illustrate the AoT distribution. **(Left)** Qualitative comparisons. All methods are able to choose a suitable position, however, both SingleStep and Where2Act cannot distinguish between actions that are moving away from or back to initial state (all directions are red) leading to inefficient exploration. In contrast, UMPNet is able to infer the correct AoT labels, hence, select the correct action to explore novel states. **(Right)** Number of unique state visited up to each step using different exploration strategy (laptop testing instances). The error bar is measure with five random seeds.

1Categories: fridge, folding chair, laptop, stapler, trashcan, microwave, toilet, window, cabinet, switch, kettle, toy, box, phone, dish washer, safe, oven, washing machine, table, kitchen pot, bucket, door.

TABLE II
GOAL CONDITIONED MANIPULATION[1]

| | Novel Instances in Train Categories | | | | | | | | | | | | Test Categories | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Inverse [30] | 0.30 | 0.21 | 0.32 | 0.31 | 0.27 | 0.17 | 0.28 | **0.09** | **0.27** | 0.25 | **0.09** | 0.34 | **0.25** | 0.32 | 0.09 | 0.17 | 0.27 | **0.15** | **0.21** | **0.00** | 0.51 | 0.27 |
| AoTOnly | 0.23 | **0.18** | 0.12 | 0.22 | 0.32 | 0.18 | 0.15 | 0.16 | 0.32 | 0.38 | 0.12 | 0.08 | 0.30 | 0.05 | 0.07 | 0.18 | 0.31 | 0.18 | 0.27 | **0.00** | 0.31 | 0.18 |
| SignedDist | 0.26 | 0.24 | 0.11 | 0.20 | 0.35 | 0.19 | 0.22 | 0.15 | 0.41 | 0.44 | 0.13 | 0.12 | 0.32 | 0.09 | 0.11 | 0.20 | 0.34 | 0.22 | 0.31 | **0.00** | 0.30 | 0.22 |
| UMPNet | **0.20** | 0.19 | **0.05** | **0.19** | **0.23** | **0.16** | **0.12** | 0.13 | 0.28 | **0.21** | 0.11 | **0.04** | 0.26 | **0.03** | **0.06** | **0.15** | **0.21** | 0.16 | 0.22 | **0.00** | **0.22** | **0.17** |
| Normalized distance to target ↓ | | | | | | | | | | | | | | | | | | | | | | |
| Inverse [30] | 0.43 | 0.68 | 0.72 | 0.55 | 0.63 | **0.89** | 0.78 | **0.65** | 0.61 | 0.52 | **0.83** | 0.54 | 0.67 | 0.59 | 0.80 | 0.73 | 0.58 | **0.83** | 0.67 | **1.00** | 0.39 | 0.68 |
| AoTOnly | 0.46 | 0.76 | 0.81 | 0.71 | 0.52 | 0.83 | 0.86 | 0.52 | 0.45 | 0.43 | 0.81 | 0.88 | 0.61 | 0.86 | **0.86** | 0.7 | 0.52 | 0.77 | 0.6 | **1.00** | 0.50 | 0.77 |
| SignedDist | 0.47 | 0.59 | 0.84 | **0.75** | 0.48 | 0.88 | 0.75 | 0.52 | 0.49 | 0.37 | 0.78 | 0.83 | 0.58 | 0.84 | 0.83 | 0.69 | 0.46 | 0.71 | 0.57 | **1.00** | 0.52 | 0.74 |
| UMPNet | **0.67** | **0.78** | **0.90** | 0.73 | **0.68** | 0.86 | **0.90** | 0.58 | **0.63** | **0.57** | 0.79 | **0.94** | **0.68** | **0.89** | **0.86** | **0.76** | **0.62** | 0.80 | **0.68** | 1.00 | **0.57** | **0.79** |
| Success rate ↑ | | | | | | | | | | | | | | | | | | | | | | |

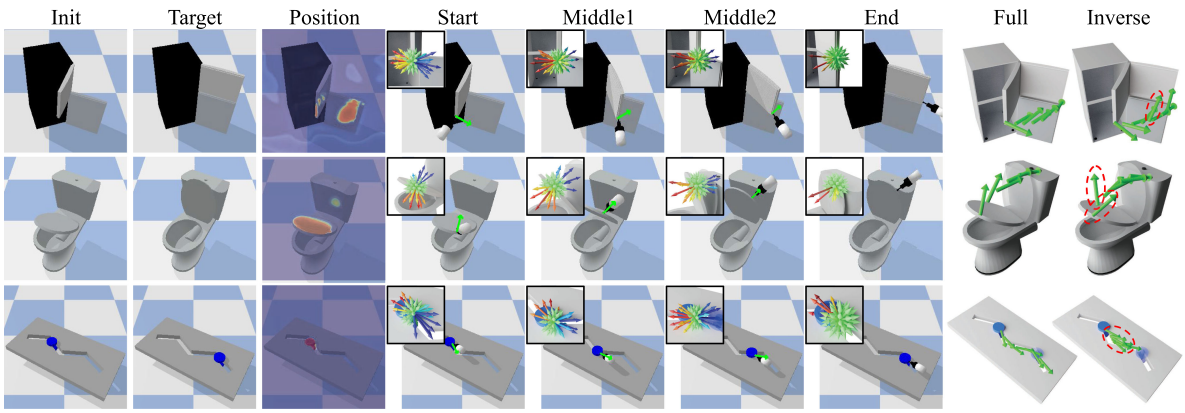


Fig. 5. **Goal conditioned manipulation results.** At the beginning or in the middle of a trajectory, the action candidates have positive (red) and negative (blue) AoT labels. To move toward the goal, the policy selects the action with the largest distance prediction and a negative AoT label (the longest blue arrow) to execute. When reaching the goal state (current and goal state are similar), the AoT labels turn non-negative for all actions since all actions will either make no change or move further away from the goal state. The [Inverse] model (right-most column) often chooses sub-optimal action directions (highlighted by red dash circles) at the beginning of the interaction sequence where the current observation is far away from the goal states.

between different action directions and choose the optimal action direction that would introduce larger state changes. As a result, [UMPNet] can achieve a better "single action effect" for all object categories, leading to more efficient state exploration when considering the entire sequence.

Effect of decomposing AoT and distance prediction: Different from [SignedDist] that directly predicts a signed distance value that combines the AoT and distance, [UMPNet] decompose its output as an AoT label (trained with classification) and a distance value (trained with regression). This decomposition helps the algorithm better disentangle these two concepts, allowing the algorithm to achieve more accurate predictions for both. As a result, [UMPNet] can achieve better performance in both metrics.

### B. Goal Conditioned Manipulation

In this experiment, we evaluate UMPNet's performance in the task of goal-conditioned manipulation. Given a target state in the form of an RGB-D image, the task is to infer a sequence of actions that manipulate the object toward the target state and halts when the object reaches the target state.

*Metrics:* The performance for this task is measured by (1) normalized distance $\mathcal{E}_{\text{goal}}$ to target state after interaction: $\mathcal{E}_{\text{goal}} = ||\vec{j}_{\text{end}} - \vec{j}_{\text{goal}}||/||\vec{j}_{\text{goal}} - \vec{j}_{\text{init}}||$, where $\vec{j}$ is vector of object's joint state. (2) success rate, where a successful case is defined as the normalized distance to the goal state is smaller than 0.1. To make the task more challenging, the initial and goal states are selected from the upper and lower limits of the joint. The initial state may be moved to ensure the task can be accomplished in 15 steps.

*Algorithm comparisons:* We compare with the [Inverse] model proposed by Agrawal *et al.* [30], a single-step inverse model for goal-conditioned manipulation. Each step takes the current and goal observation as input and predicts the action that would change the state from the current state to the goal state. This model is trained on the same state-action pairs $(s_t, s_{t+1}, a_t)$ as our method, and the action output is trained with direct regression loss.

*Results and analysis:* Table II shows that comparing to prior works [Inverse] and other alternative approaches, [UMPNet] is able to achieve more precise goal-conditioned manipulations by moving the object to a state that is closer to the target (lower $\mathcal{E}_{\text{goal}}$ value). From the qualitative comparisons in Fig. 5 we can observe that the performance of the [Inverse] model is much worse at the beginning of the interaction, where the algorithm often selects sub-optimal action directions that make less progress towards the goal (actions highlighted in red dash circle). Since the [Inverse] model only takes consecutive observations as input during training, it struggles

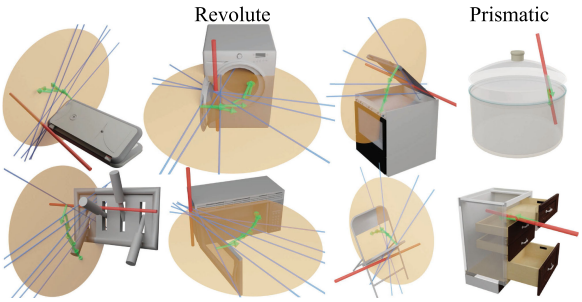


Fig. 6. **Action → Articulation.** The joint axes (red) are inferred from the actions selected by the learned policy (green), which indicates the system's implicit understanding about the objects' articulation structure.

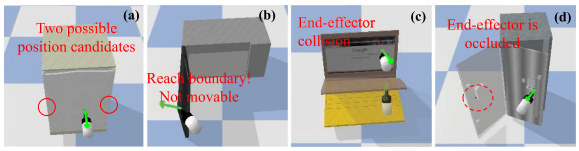


Fig. 7. Typical failure cases.

to handle long-horizon manipulation tasks, where the current observation is far away from the goal states. Similar to exploration experiments, we observe that [AoTOnly] often chooses sub-optimal action direction as it is unaware of the actual magnitudes (i.e., distance) of different action effects.

### C. Inferring Objects' Articulation Structure From Interactions

We hypothesize that one of the requirement for learning a universal policy for *any* articulated object is the ability to understand the object's underlying articulation structure and how this structure react to different actions. Hence, the action selected by the policy should also, in return, reflects its belief on the objects' structure. For example, we often apply forces along the axis for prismatic joints while applying actions perpendicular to the rotation axis for revolute joints.

To visualize the policy's implicit belief about the object's structure, we compute the joint parameters inferred from the actions selected by the policy. To compute the prismatic joint, we simply take the average of the action directions. To compute the revolute joint, we first compute a common action plane in the 3D space (brown plane in Fig. 6). The normal direction of the plane $\vec{n} \in R^3$ is chosen as $\min_{||\vec{n}||=1} \sum_{t=1}^{T} |\vec{n} \cdot a_t|$, where $a_t$ is the action direction in each interaction step. Then we vote for the axis position by computing the interaction sections between the directions perpendicular to all the actions in the common plane (blue lines in Fig. 6). Finally, the final axis position is voted among the intersection points between each pair of the perpendicular lines. Fig. 6 shows examples of inferred joint parameters for objects with different articulation structures (red lines).

We also quantitatively evaluate the inferred joint parameters. While the algorithm has never been supervised on any of the joint parameters, it is able to estimate the joint axis orientation with an average error $< 11.6^\circ$ for revolute joints and $< 32.2^\circ$ for prismatic joints. Note that the error in prismatic joint estimation is higher since these objects often has higher tolerance on the sub-optimal action directions.

### D. Real-World Experiment

Finally, we validate our method on a real-world platform with a calibrated RGB-D camera (Intel RealSense D415), a UR5 robot, and a suction gripper. Fig. 8(a) shows the real-world setup. In this experiment, we directly tested UMPNet trained in simulation on four different objects – box, laptop, microwave, and stapler. The inferred action trajectories to open and close the microwave are shown in Fig. 8(b). The qualitative result of goal-conditioned manipulation shown in Fig. 8(c) demonstrates that the trained model is able to infer proper grasping positions and action directions for different objects and goal conditions. While performing large-scale real-world training for UMPNet can still be challenging, we believe these results demonstrate the promises of the proposed method in real-world applications. We observed that there are a few real2sim gaps that could impact real-world performance. For example, the noise captured by the depth camera could affect direction inference. For objects don't have a fixed base (e.g., microwave), they might experience unexpected movements during interactions, and therefore negatively impact the algorithm performance. In addition, our policy doesn't consider real robot situation, for example, whether the grasping position can be reached by a real robot, the moving trajectory is safe, the grasping surface is flat enough for a robust suction. All these issues about real robot platforms should be considered in our policy in future works.

### E. Limitations and Failure Cases

*Assumptions:* To allow goal-conditioned manipulation with reversed AoT actions, we assume the action trajectories are bi-directional in time (i.e., they are valid in either direction). While this assumption is true for most articulated objects, it does not apply to irreversible actions such as gluing or locking. In addition, our system assumes the agent uses a suction-based end-effector, which can provide robust grasps for a large variety of objects and is widely used in many real-world robotics systems. However, the policy cannot generalize to other grippers that requires more precise grasp poses. Finally, our system assumes there is only a single articulated object with 1 DoF prismatic or revolute joint on a planar surface, and the goal state can only be input as an image with the same scene.

*Failure Cases:* Fig. 7 case (a) is ambiguous in position selection since the door could be opened from both sides, where the policy chooses to drag the middle of the door. In case (b), the selected action can't change the object state since the microwave's door reaches boundary. However, the joint range can't be easily inferred from observation since some microwaves can be opened up to 180°. In case(c), policy infers actions that will cause collisions between the end-effector and the object. In case(d), the end-effector is occluded after interactions. While a human is able to change the viewpoint for better observation, our agent uses a fixed camera position and therefore not robust for occlusion. Both (c) and (d) could be addressed by better modeling the agent's embodiment including end-effector and camera placement.

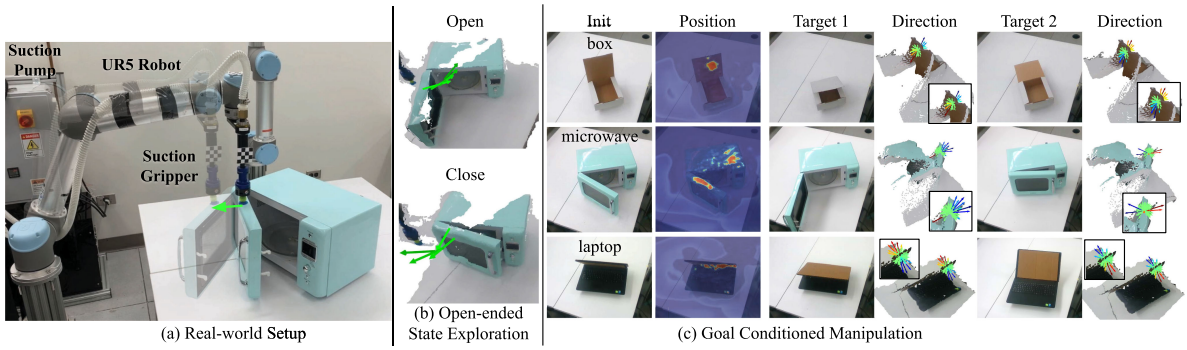


Fig. 8. **Real-world experiment.** We test the model trained in simulation on a real-world platform. **(a)** We an RGB-D camera to capture visual observation and a UR5 with a suction gripper for manipulation. **(b)** Action trajectory. **(c)** For each object, we visualize the inferred action position and direction for two different target states. To move toward the goal, the policy will select the action with the largest distance prediction and a negative AoT label (the longest blue arrow) to execute.

## V. Conclusion

We introduce the Universal Manipulation Policy Network (UMPNet) – a single image-based policy network that infers closed-loop action sequence for manipulating articulated objects. The policy is trained with self-guided exploration without human demonstrations, scripted policy, or pre-defined goal conditions. Our experiment results demonstrate that the learned policy is able to perform well in both open-ended exploration and goal-conditioned manipulation and outperforms alternative approaches in both tasks.

## Acknowledgment

The author would like to thank Google for the UR5 robot hardware. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

## References

[1] S. Song, A. Zeng, J. Lee, and T. Funkhouser, "Grasping in the wild: Learning 6DoF closed-loop grasping from low-cost demonstrations," *IEEE Robot. Autom. Lett.*, vol. 5, no. 3, pp. 4978–4985, Jul. 2020.

[2] J. K. Li, W. S. Lee, and D. Hsu, "Push-net: Deep planar pushing for objects with unknown physical properties," in *Proc. Robot.: Sci. Syst.*, 2018.

[3] A. Zeng, S. Song, J. Lee, A. Rodriguez, and T. Funkhouser, "TossingBot: Learning to throw arbitrary objects with residual physics," in *Proc. Robot.: Sci. Syst.*, 2019.

[4] E. Klingbeil, A. Saxena, and A. Y. Ng, "Learning to open new doors," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2010, pp. 2751–2757.

[5] B. Abbatematteo, S. Tellex, and G. Konidaris, "Learning to generalize kinematic models to novel objects," in *Proc. Conf. Robot Learn.*, 2019, pp. 1289–1299.

[6] A. Jain, R. Lioutikov, C. Chuck, and S. Niekum, "ScrewNet: Category-independent articulation model estimation from depth images using screw theory," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2021, pp. 13670–13677.

[7] M. Mittal, D. Hoeller, F. Farshidian, M. Hutter, and A. Garg, "Articulated object interaction in unknown scenes with whole-body mobile manipulation," 2021, *arXiv:2103.10534*.

[8] A. Jain and S. Niekum, "Learning hybrid object kinematics for efficient hierarchical planning under uncertainty," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2020, pp. 5253–5260.

[9] T. Rühr, J. Sturm, D. Pangercic, M. Beetz, and D. Cremers, "A generalized framework for opening doors and drawers in kitchen environments," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2012, pp. 3852–3858.

[10] T. Harada, A. Tejero-de Pablos, S. Quer, and F. Savarese, "Service robots: A unified framework for detecting, opening and navigating through doors," in *Proc. Int. Conf. Softw. Technol.*, 2019, pp. 179–204.

[11] A. J. Schmid, N. Gorges, D. Goger, and H. Worn, "Opening a door with a humanoid robot using multi-sensory tactile feedback," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2008, pp. 285–291.

[12] C. C. Kessens, J. B. Rice, D. C. Smith, S. J. Biggs, and R. Garcia, "Utilizing compliance to manipulate doors with unmodeled constraints," in *Proc. IEEE/RJS Int. Conf. Robot. Autom.*, 2010, pp. 483–489.

[13] S. Y. Gadre, K. Ehsani, and S. Song, "Act the part: Learning interaction strategies for articulated object part discovery," in *Proc. Int. Conf. Comput. Vis.*, 2021.

[14] X. Li, H. Wang, L. Yi, L. Guibas, A. L. Abbott, and S. Song, "Category-level articulated object pose estimation," 2019, *arXiv:1912.11913*.

[15] S. Niekum, S. Chitta, A. G. Barto, B. Marthi, and S. Osentoski, "Incremental semantically grounded learning from demonstration," in *Proc. Robot.: Sci. Syst.*, 2013.

[16] J. Kober, B. Mohler, J. Peters, and O. Sigaud, "Imitation and reinforcement learning for motor primitives with perceptual coupling," *From Motor Learn. Interaction Learn. Robots*, vol. 264, no. 2010, pp. 209–225, 2010.

[17] P. Pastor, H. Hoffmann, T. Asfour, and S. Schaal, "Learning and generalization of motor skills by learning from demonstration," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2009, pp. 763–768.

[18] P. Sermanet, C. Lynch, J. Hsu, and S. Levine, "Time-contrastive networks: Self-supervised learning from multi-view observation," CoRR, 2017.

[19] D. Huang *et al.*, "Neural task graphs: Generalizing to unseen tasks from a single video demonstration," 2018, *arXiv:1807.03480*.

[20] D.-A. Huang *et al.*, "Motion reasoning for goal-based imitation learning," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2020, pp. 4878–4884.

[21] T. Zhang, Z. McCarthy, O. Jow, D. Lee, K. Goldberg, and P. Abbeel, "Deep imitation learning for complex manipulation tasks from virtual reality teleoperation," 2017, *arXiv:1710.04615*.

[22] C. Lynch *et al.*, "Learning latent plans from play," in *Proc. Conf. Robot Learn.*, 2020, pp. 1113–1132.

[23] T. Nagarajan, C. Feichtenhofer, and K. Grauman, "Grounded human-object interaction hotspots from video," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, 2019, pp. 8687–8696.

[24] T. Nagarajan, Y. Li, C. Feichtenhofer, and K. Grauman, "Ego-topo: Environment affordances from egocentric video," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2020, pp. 160—169.

[25] S. Brahmbhatt, C. Ham, C. C. Kemp, and J. Hays, "ContactDB: Analyzing and predicting grasp contact via thermal imaging," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 8709–8719.

[26] K. Mo, L. Guibas, M. Mukadam, A. Gupta, and S. Tulsiani, "Where2Act: From pixels to actions for articulated 3D objects," in *Proc. Int. Conf. Comput. Vis.*, 2021.

[27] S. Nasiriany, V. H. Pong, S. Lin, and S. Levine, "Planning with goal-conditioned policies," 2019, *arXiv:1911.08453*.

[28] F. Xiang *et al.*, "SAPIEN: A simulated part-based interactive environment," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2020, pp. 11097–11107.

[29] E. Coumans and Y. Bai, "Pybullet, A python module for physics simulation in robotics, games and machine learning," 2017. [Online]. Available: http://pybullet.org

[30] P. Agrawal, A. V. Nair, P. Abbeel, J. Malik, and S. Levine, "Learning to poke by poking: Experiential learning of intuitive physics," in *Proc. 30th Int. Conf. Neural Inf. Process. Syst.*, 2016, pp. 5074–5082.